Dynamic Stylized Shading Primitives

David Vanderhaeghe, Romain Vergne, Pascal Barla, William Baxter

Dynamic Stylized Shading Primitives

David Vanderhaeghe*
IRIT - Université de Toulouse

Romain Vergne
University of Giessen

Pascal Barla
INRIA

William Baxter†
Google

Abstract

Shading appearance in illustrations, comics and graphic novels is designed to convey illumination, material and surface shape characteristics at once. Moreover, shading may vary depending on different configurations of surface distance, lighting, character expressions, timing of the action, to articulate storytelling or draw attention to a part of an object. In this paper, we present a method that imitates such expressive stylized shading techniques in dynamic 3D scenes, and which offers a simple and flexible means for artists to design and tweak the shading appearance and its dynamic behavior. The key contribution of our approach is to seamlessly vary appearance by using a combination of shading primitives that take into account lighting direction, material characteristics and surface features. We demonstrate their flexibility in a number of scenarios: minimal shading, comics or cartoon rendering, glossy and anisotropic material effects; including a variety of dynamic variations based on orientation, timing or depth. Our prototype implementation combines shading primitives with a layered approach and runs in real-time on the GPU.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture;

Keywords: non-photorealistic rendering, stylized shading, shape depiction, real-time rendering

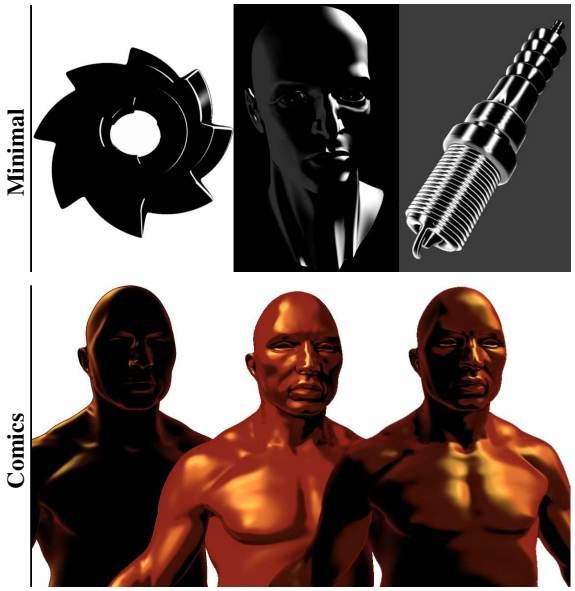


Figure 1: ***Shading styles.*** *Various styles are created by combining multiple shading primitives.*

1 Introduction

Shading in computer graphics is generally designed to simulate reality by using a combination of complex materials and lighting environments. Although this approach has many applications in architecture or special effects for movies, other applications rather use shading in a stylized manner. Examples of stylized shading choices abound in artistic illustrations and graphic novels [Hogarth 1991; McCloud 1994]. Even if such illustrations may make use of very different media such as watercolor, oil paint, acrylic or pencil for instance, they all tend to reproduce shading in similar respects. Firstly, they are not constrained by physical accuracy: a few simple shading gradients are enough to produce a convincing appearance, and sharp color transitions are often used to create cartoon effects. More importantly, they convey illumination, material and surface shape characteristics *all at once*. Shading is thus often used to articulate storytelling, by exaggerating facial expressions through the enhancement of reflections and surface details, or by accentuating character silhouettes via sharp rim lighting.

There is another advantage in relaxing physical constraints though, as illustrated in the work of Hogarth [1991]: it allows artists to create expressive compositions, whereby shading takes on different appearances with different configurations of surface distance, lighting, character expressions, timing of the action, *etc*. This is especially noticeable in graphic novels where, from one frame to another, shading appearance may change significantly.

Turning hand-made artworks that exhibit stylized shading appearance into computer-based animations is not trivial. An important requirement is to provide artists with a direct control over surface shape depiction, material characteristics, lighting directions and stylization effects through time. An obvious approach would be to let users paint and animate shading directly onto 3D objects, but it would require a tremendous amount of hand-tuning and keyframing. The alternative solution that we investigate in this paper is to automate common shading behaviors, while providing sufficient access to artistic control. With this approach, dynamic scene changes raise the additional challenge of seamlessly transforming a shading appearance into another in real-time. As explained in Section 2, existing shading techniques are often limited in the handling of dynamic shading changes; moreover, they are usually restricted to specific stylization effects.

In contrast, our goal is to provide a single approach to create various dynamic stylized shading effects. Our solution is to decompose shading into a set of *procedural primitives* amenable to dynamic behaviors. Each primitive is controlled by a single light source, and its appearance is determined by material characteristics, stylization choices, and view-dependent surface features. We make use of a single shading primitive model in order to permit continuous changes of material, shape and stylization effects. The design choices that have lead to our model are presented and discussed throughout Section 3. Primitives are evaluated and composited in real-time using the GPU, and a few primitives are sufficient to mimic most common shading styles, as demonstrated in Section 4. We discuss their benefits, limitations and future extensions in Section 5.

*e-mail: vdh@irit.fr

†Work completed in part at OLM Digital, Inc.

2 Previous work

One of the earliest forms of stylized shading is *Toon Shading*, which simply segments the result of a diffuse shading function into a few color bands. The *Technical Illustration Shader* of Gooch et al. [1998] uses a similar approach with an unclamped diffuse term (also called *Half-Lambertian*), and cool-to-warm color gradients. Alternative color gradients stored in 1D textures have been used in video games [Mitchell et al. 2007] to reproduce illustrative shading styles.

More complex effects can be obtained with 2D color gradients. The *Lit Sphere* of Sloan et al. [2001] consists of a single shaded sphere stored in a 2D texture and looked up using screen-space normals. Variations on this basic idea have been successfully used in different contexts such as volumetric rendering [Bruckner and Gröller 2007] or digital sculpture software (e.g., ZBrush® or MudBox®). Although the Lit Sphere provides a very natural approach to the creation of stylized shading, it produces a relatively static shading appearance, since the lighting direction is implicitly "baked" in the 2D texture. The *X-Toon* method of Barla et al. [2006] takes a different approach where the 1D toon function is extended to a 2D function, stored in a texture. The additional dimension is used to create effects that vary with depth or surface orientation for instance. Although light direction may vary in this approach, it permits only one effect that is baked in each 2D texture.

An alternative approach to stylized shading is to take advantage of inverse rendering techniques that allow direct material and illumination editing. The *Illumination Brush* of Okabe et al. [2007] permit such control: starting from a known BRDF, they infer the environment lighting by directly painting on the object the desired result of either diffuse or specular reflections. Even if the paint strokes may be moved around for stylization purpose, the adaptation of the method to the creation of stylized shading is not straightforward, since it relies on BRDF models and realistic illumination. We rather seek a method that is not bound by such physical constraints.

One reason for providing additional freedom is to give artists the ability to control how shape is conveyed through shading. Previous techniques have proposed extensions to existing shading models that enhance surface characteristics. The *Exaggerated shading* technique of Rusinkiewicz et al. [2006] locally aligns light directions with grazing surface angles so that details are revealed through variations of a Half-Lambertian shader. Downsides of this approach are that it tends to produce "flat" appearances where the main illumination direction is lost, and visual artifacts may occur during light motion. The *Apparent Relief* technique [Vergne et al. 2008] rather exploits the additional dimension of the X-Toon shader to convey shape features computed via a combination of object- and image-space measurements. The method is able to depict shape features at varying scales, but exhibits artifacts with distant objects due to the combination of surface measurements. Both methods are restricted to a single shading style though. This is not the case with the *Radiance Scaling* technique [Vergne et al. 2010], which works with arbitrary shading models: it consists in scaling incoming radiance according to surface and material characteristics. However, because incoming light directions are treated independently, stylized shading gradients are likely to be disrupted.

Apart from the simplest stylized shading techniques, the methods presented so far do not permit any dynamic control over stylized material characteristics. The cartoon highlights of Anjyo et al. [2003; 2006] deal with the specific case of the Blinn-Phong specular term. Highlight shape is tweaked via translation, rotation, scaling, splitting, squaring, and Boolean operators applied to the half vector. Artists can control highlight shape deformations through keyframing. Pacanowski et al. [2008] propose a more direct interface whereby artists draw the 2D profile of a highlight in a plane perpendicular to the reflection direction. In their system, highlight shape can be deformed based on a few light directions to control shading appearance at grazing angles. The two methods focus essentially on highlight shape, whereas other material characteristics are left unchanged, and shape features are ignored.

In contrast to previous work, our approach provides a single solution to the stylized shading of 3D scenes where both surface features and material characteristics are conveyed dynamically.

3 Shading model

In our approach, stylized shading is designed as a combination of a few procedural shading primitives (Section 3.1) whose parameters are dynamically modified when applied to 3D objects (Section 3.2). Parameters of our shading model are summarized in Table 1.

Symbol	Domain	Name
$K(\cdot)$	$[0,\pi] \rightarrow [0,1]^3$	color
α	$[0,1]$	specularity
τ	$[-\pi,\pi]$	extent
f	$[0,\pi]$	intensity fall-off
c	$[f,\pi]$	intensity cut-off
λ	$(-1,1)$	material anisotropy
μ	$\mathbb{R}$	surface enhancement
χ	$\mathbb{R}$	concave/convex transition

Table 1: *List of shading parameters.*

3.1 Shading primitives

Three principal requirements have led to the design of our shading primitives. First, a single primitive model should account for different shading behaviors, from diffuse to specular. Second, continuous changes of primitive parameters should lead to continuous (and ideally perceptually uniform) visual changes. Third, a primitive should be easily manipulated directly on 3D models, and in particular it should not require the tweaking of too many parameters. With these technical goals in mind, we define a shading primitive as composed of three basic ingredients: an angular parametrization u, and a pair of color and intensity profile functions, defined on this parametrization and noted K and I respectively.

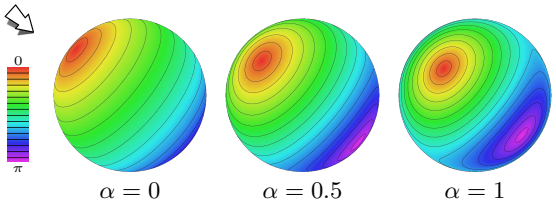
The parametrization $u : \mathbb{S}^2 \times \mathbb{S}^2 \times \mathbb{S}^2 \rightarrow [0,\pi]$ is defined by

$$u(\boldsymbol{n},\boldsymbol{l},\boldsymbol{v}) = \lfloor \mathrm{acos}(\boldsymbol{d}_\alpha \cdot \boldsymbol{l}) - \tau \rfloor, \tag{1}$$

where $\boldsymbol{n}$, $\boldsymbol{l}$ and $\boldsymbol{v}$ correspond to the normal, light and view vectors, $\boldsymbol{d}_\alpha$ is a reference direction that controls how the parametrization evolves when $\boldsymbol{l}$ is rotated, and $\lfloor x \rfloor$ is a function that clamps x to $[0,\pi]$. The user-specified parameter $\tau \in [0,\pi]$ permits to control the extent of the shading primitive independently of its color and intensity profiles. The reference direction $\boldsymbol{d}_\alpha$ is given by

$$\boldsymbol{d}_\alpha = \frac{(1-\alpha)\,\boldsymbol{n} + \alpha\,\boldsymbol{r}}{||(1-\alpha)\,\boldsymbol{n} + \alpha\,\boldsymbol{r}||}, \tag{2}$$

where $\boldsymbol{r} = 2(\boldsymbol{v}\cdot\boldsymbol{n})\,\boldsymbol{n} - \boldsymbol{v}$ is the view vector reflected around the surface normal, and α is a user-controlled parameter that interpolates between diffuse ($\alpha = 0$) and specular ($\alpha = 1$) shading behaviors. Different choices of parametrization are illustrated in Figure 2: we set up an orthographic view and render a sphere that represents the entire set of normal directions for a particular light direction $\boldsymbol{l}$. The angular parametrization, displayed with a color code and isolines, evolves continuously when $\boldsymbol{l}$, τ or α are manipulated.



$\alpha = 0$ $\alpha = 0.5$ $\alpha = 1$

Figure 2: ***Shading parametrization****: Here we illustrate the parametrization u for the light direction shown at top left. The left most color bar presents the color code used. By varying α, different shading behaviors are continuously produced from diffuse ($\alpha = 0$) to specular ($\alpha = 1$). We have used an offset of $\tau = 0$ in all cases.*

Once a parametrization has been chosen, it becomes easier to specify color and intensity profiles. In our system, the color profile $K(u)$ is simply defined as a smooth 1D color gradient (commonly called a color ramp), while intensity is controlled by a more complex profile function. This choice is motivated by the observation that sharp transitions due to either shininess or cartoon effects are more easily controlled through a scalar function.

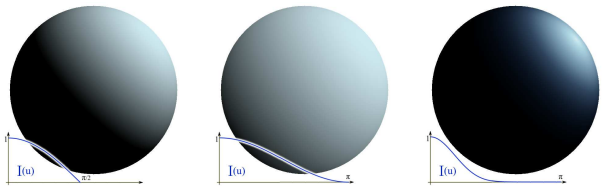
The intensity profile $I : [0, \pi] \rightarrow [0, 1]$ is defined by

$$I(u) = |\beta + (1 - \beta)\cos u)|_+^\gamma \,, \tag{3}$$

where $\beta \in [-1/2, 1/2]$ is a bias parameter that permits to extend the primitive intensity toward the interval $[\pi/2, \pi]$, $\gamma \in \mathbb{R}^+$ is an exponent parameter that determines the intensity fall-off rate, and $|x|_+$ is a function that clamps negative values of x to 0. Figure 3 shows how different values of β and γ allow users to control shading appearance for a fixed parametrization.

The bias and exponent parameters are inspired respectively by the Half-Lambert and Phong specular terms, which have been frequently used in previous work on stylized shading. In our system though, bias and exponent are employed in all situations, not only because we use a single primitive model for both diffuse and specular effects, but also because it provides additional creative freedom. This flexibility has one minor drawback though: specular primitives produce exaggerated rim effects when the light direction comes from the back of the object because our formulation does not incorporate the geometric term $(\boldsymbol{n}\cdot\boldsymbol{l})$ in this case. We retain this rim effect because it offers an interesting specular style, but the original Phong-like behavior may optionally be re-introduced by multiplying $I(u)$ by $(1 - \alpha) + \alpha(\boldsymbol{n} \cdot \boldsymbol{l})$ (see Supplemental material).

Uniformly changing bias and exponent values leads to sudden changes in shading appearance as shown for γ at the top of Figure 4. This is not only a matter of user interface; it also raises issues during interpolation such as when applying the dynamic shading variations presented in Section 4. A better choice of parameters are the fall-off $f \in [0, \pi]$ and cut-off $c \in [f, \pi]$ angles at which the intensity profile respectively exhibits an inflection and reaches 0. For instance, uniformly changing f values leads to a more uniform interpolation as shown at the bottom of Figure 4. We explain how to compute bias and exponent based on these parameters in Equations 7 and 8 of the Appendix. Classic shading behaviors are then easily reproduced: Phong's diffuse term corresponds to $f = c = \pi/2$, and the Half-lambertian term is obtained by further setting $c = \pi$; Phong's specular term (with the geometric term re-introduced) is the same as diffuse, but leaves f free to vary in the $[0, \pi/2]$ range; cartoon shading occurs when $c = 0$ and $\tau > 0$, and ambient shading when $\tau = \pi$. In our system, τ, f and c are directly controllable on top of 3D objects via isolines.



$\beta = 0,\ \gamma = 1$ $\beta = 0.5,\ \gamma = 1$ $\beta = 0.5,\ \gamma = 7$

Figure 3: ***Shading intensity****: by varying β and γ, different shading appearances are continuously obtained from sharp to smooth profiles. Here we use a simple diffuse shading parametrization ($\alpha = 0,\ \tau = 0$) and a smooth color ramp in the right image.*

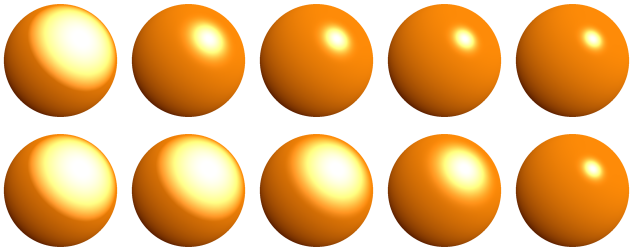


Figure 4: ***Profile interpolation****: changing the exponent γ from 1 to 30 by uniform increments exhibits non-uniform shading appearance interpolation (top row). In contrast, with uniform increments of f, keeping $c = \pi/2$, we obtain an interpolation behavior that is more uniform (bottom row).*

3.2 Shading variations

When applied to a 3D object, a shading primitive is evaluated at each surface point by computing local light and view vectors as in traditional shading techniques. A single directional or point light source is required for each primitive. Shading variations automatically occur from one surface point to another when using a point light source and/or a perspective camera. However, since primitives are entirely procedural, their parameters may also be dynamically modified across the object to create *intentional* shading variations. We distinguish between two types of variations: those that only depend on local object characteristics (such as shape or material features), and those that depend on scene or shot characteristics (such as depth, timing of action, etc). The former is presented in the remainder of this Section while the latter is presented in Section 4 when we consider the combination of multiple of our shading primitives.

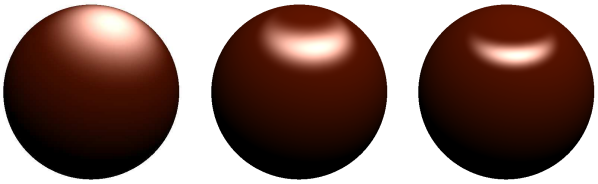
The main source of real-world local material variation is caused by anisotropic distributions of microfacets. They are typical of materials such as brushed metals or satin, and usually controlled by a user-specified tangent field defined at each surface point. To mimic such material effects, we adapt the solution of Anjyo et al. [2003; 2006] to work with our primitive model. We first compute the half vector $\boldsymbol{h}$ between $\boldsymbol{l}$ and $\boldsymbol{v}$, then transform it based on the local tangent $\boldsymbol{t}$ and bi-tangent $\boldsymbol{b}$ vectors as in the following:

$$\tilde{\boldsymbol{h}} = \frac{s_\lambda \boldsymbol{h_t} + \frac{1}{s_\lambda}\boldsymbol{h_b} + \boldsymbol{h_n}}{||s_\lambda \boldsymbol{h_t} + \frac{1}{s_\lambda}\boldsymbol{h_b} + \boldsymbol{h_n}||} \tag{4}$$

where $\boldsymbol{h_{t,b,n}}$ are the component vectors of $\boldsymbol{h}$ in tangent space (e.g., $\boldsymbol{h_t} = (\boldsymbol{h} \cdot \boldsymbol{t})\boldsymbol{t}$) and s_λ is a scaling factor defined by

$$s_\lambda = \begin{cases} 1 - \eta + \eta\frac{1}{1-\lambda} & \text{if } \lambda \geq 0 \\ \left(1 - \eta + \eta\frac{1}{1+\lambda}\right)^{-1} & \text{if } \lambda < 0 \end{cases} \tag{5}$$

with $\lambda \in (-1, 1)$ a user-controlled anisotropy parameter, and $\eta = (\boldsymbol{l} \cdot \boldsymbol{v} + 1)/2$ a *back-lighting* term. Intuitively, Equation 4 consists



$\alpha = 0$ $\alpha = 0.5$ $\alpha = 1$

Figure 5: ***Shading anisotropy****: the λ parameter controls anisotropy (here we use $\lambda = -0.5$) and is progressivelly introduced when the shading behavior tends toward specular.*

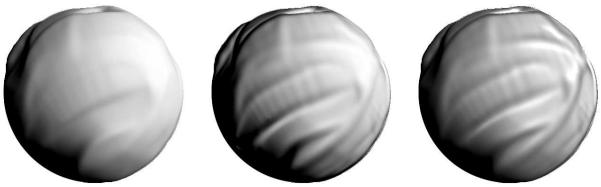
in rotating $\boldsymbol{h}$ toward or away from $\boldsymbol{n}$, depending on its projection in the tangent plane: when $\lambda > 0$, half vectors are compressed in the $\boldsymbol{t}$ direction; when $\lambda < 0$, half vectors are stretched toward the $\boldsymbol{t}$ direction; and when $\lambda = 0$, half-vectors are kept unchanged. We systematically apply the inverse scaling, $1/s_\lambda$, in the direction of $\boldsymbol{b}$ to roughly maintain highlight size regardless of anisotropy as seen in Equation 4. The back-lighting term η in Equation 5 is needed to revert to an isotropic behavior ($s_\lambda = 1$) when the light direction is opposite to the view direction, since in this case the half-vector is ill-defined.

Our shading model does not produce specular behaviors based on the half-vector $\boldsymbol{h}$ but rather makes use of the reflected view vector $\boldsymbol{r}$ for interpolation purpose. We thus first reflect $\boldsymbol{l}$ around $\tilde{\boldsymbol{h}}$ to compute the transformed view vector $\tilde{\boldsymbol{v}} = 2(\boldsymbol{l} \cdot \tilde{\boldsymbol{h}})\tilde{\boldsymbol{h}} - \boldsymbol{l}$, and then reflect $\tilde{\boldsymbol{v}}$ around $\boldsymbol{n}$ to obtain $\tilde{\boldsymbol{r}} = 2(\boldsymbol{n} \cdot \tilde{\boldsymbol{v}})\boldsymbol{n} - \tilde{\boldsymbol{v}}$. As illustrated in Figure 5 where the tangent field is aligned with sphere meridians, anisotropy is ignored in the diffuse case ($\boldsymbol{d}_0 = \boldsymbol{n}$), and appears progressively when tending toward a specular behavior ($\boldsymbol{d}_1 = \tilde{\boldsymbol{r}}$).

Another important set of local object characteristics are surface features, and recent techniques have provided solutions for enhancing them through shading. We present a novel approach where we manipulate directly the shape of shading primitives instead of modifying independent shading values as in previous work. We first measure surface curvature κ at each visible surface point. Although our approach is independent of the choice of curvature measure, we have chosen the view-centered mean curvature of Vergne et al. [2010] because it provides automatic levels-of-detail and is computed dynamically from screen-space normals. We then offset the parametrization function based on local surface curvature:

$$\tilde{\tau} = \tau + \mu \tanh(\kappa\chi), \tag{6}$$

where $\mu \in \mathbb{R}$ controls the magnitude of enhancement and $\chi \in \mathbb{R}$ the slope of the transition between concave ($\kappa < 0$) and convex ($\kappa > 0$) features. The reason why we have chosen to vary τ is that it only modifies the *extent* of a primitive: for instance, shading may be "attracted" in convexities and "repelled" from concavities. In comparison, other alternatives such as Radiance Scaling modify shading appearance by disrupting existing shading gradients as shown in Figure 6.



Original Proposed approach Radiance Scaling

Figure 6: ***Shape depiction****: primitive extent is modified to better depict surface features. Here, the shading over a blobby object (left) is modified to convey concave features using $\mu = 0.3$ and $\chi = 30$ (middle). Compared to Radiance Scaling (right), our solution better preserves shading gradients while still conveying shape.*

We have also tried blending between a pair of color ramps inside a single primitive, using a similar transition function for concave to convex regions. This technique is inspired by the ZBrush® MatCaps, where two LitSpheres are blended with a concave-to-convex transition function. An advantage of our approach is that shading primitives may have different blending transitions, while in the case of MatCaps, the entire LitSpheres are blended together. However, this technique strongly modifies material appearance, and is thus only used for specific effects (see supplemental material).

4 Results

Dynamic stylized shading primitives are currently implemented in GLSL and we obtain real-time performance (> 60 fps on a NVIDIA GTX480) in 1600×1200 for all the examples in this paper. Primitive evaluation is performed in a pixel shader, and we have chosen a layered interface for combining primitives in a way similar to classic image-processing software such as Photoshop® or Gimp®. As demonstrated in all our results and the supplemental video, a few primitives combined with alpha or additive blending permit to quickly imitate existing artwork or mimic complex materials. This is not only valuable in interactive applications, but also at the compositing stage every time light-weight re-lighting is preferable to a more heavy-weight solution at the 3D rendering stage. Our shader is given in the Appendix.

The *tutorial* video illustrates the process of creating a stylized shading in our prototype system. It demonstrates various features: shading behaviors, isoline controls, shape-based primitive extent and color variations, primitive blending, etc. Each primitive position is controlled independently to create ambient, diffuse, highlights and rim effects. We have also found it useful to bind all light sources to a common illumination reference frame. This is the solution we have used for most of the results shown in the *gallery* video.

Some of these shading styles are presented in Figures 1 and 7. A few primitives are sufficient to obtain compelling *minimal* shading results. In the first example, we use a single white specular primitive deformed to convey convex shape features in a quite exaggerated way, which provides a compelling depiction of shape within a single layer. In the second example, we use a pair of opposite specular primitives deformed so that all surface details of the face are represented. The third example makes use of 3 anisotropic primitives that convey a metallic appearance to the model. The *comics* shading example shows a character rendered with 7 primitives and lit from three different directions. Shape features are used to align shading with anatomical details and to darken colors in concavities. Simpler primitives are used for *cartoon* shading styles (we have used 6 and 4 primitives for the left and right images respectively). Embossing and contouring effects are created by attaching the light source to the camera, while sharp transitions are obtained when setting $c = 0$ and $\tau > 0$. More complex *glossy* effects are obtained by combining multiple primitives. We have used 6 primitives in both images of the dog model; they differ only in color and light source positions. The same shape-based variations have been applied to demonstrate that we obtain similar surface depictions in both cases. The Venus and Buddha images illustrate *anisotropic* shading styles, where we have combined 4 primitives in both cases to mimic velvet and gold.

Intentional and dynamic variations of primitive parameters prove to be useful in many situations, as shown in Figure 8. For instance, varying primitive profile and extent based on object *orientation* is useful in scientific illustration scenarios. In the sanguine-like shading, the reddish and whitish shading primitives are modified when seen from behind, while a third ambient primitive is kept constant. In the beetle image (made of 7 primitives), the highlight profiles and their blending are altered to draw attention to the material when on

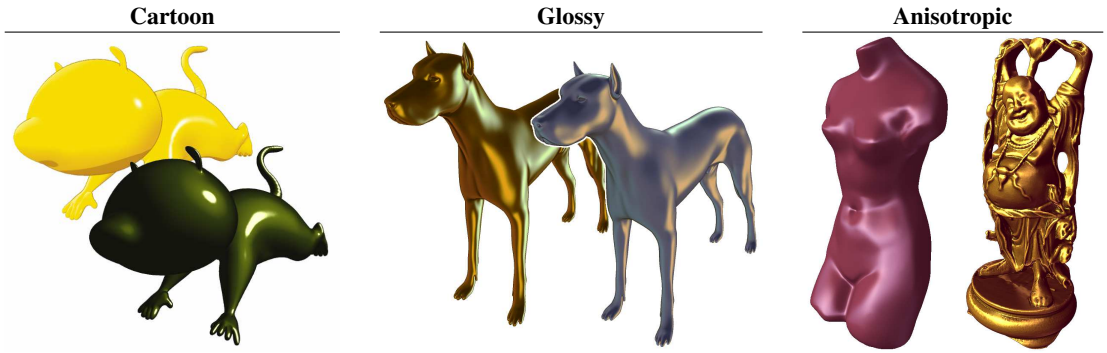


Figure 7: ***Shading styles.*** *Various styles are created by combining multiple shading primitives.*

top of the insect, while the bottom surface is depicted with exaggerated shape-based deformations of a diffuse primitive. A more direct shading artistic control is obtained through *key-framing*, as in the elephant image which shows two frames of an animation where shape-based primitive deformations are activated each time the elephant hits the ground. Although we have applied the effect everywhere on the elephant, it could be easily localized by means of an alpha map. Finally, continuous variations are obtained by mapping any scene parameter to one or more primitive parameters. An example is given in the landscape image where an aerial perspective effect is produced by mapping *depth* to shading primitive variations. Here, we alter the color of a diffuse primitive while smoothly suppressing its shape-based deformations with increasing depth; we also let another slightly specular primitive only appear in the foreground to give more volume to nearby relief. Note that using a simpler approach such as X-Toon [Barla et al. 2006] would only permit one depth-based effect at a time.
With our prototype system, it takes a couple minutes to get a first shading result, and up to half an hour for refining more complex styles. Although we have shown quite simple dynamic controls based on depth, orientation and keyframing, our shading primitives may be controled in more complex ways, which ultimately depend on the final application.

5 Discussion

We have presented an approach for dynamic stylized shading based on the composition of a few procedural primitives. Our solution is well adapted to both interactive applications such as video games, scientific illustration, or digital sculpture; and offline applications such as stylized rendering and compositing for animated films and commercials. Our shading model is designed to be continuously interpolated among a variety of stylized shading effects, such as diffuse or specular behaviors, surface and anisotropy features, etc.

A limitation of our stylized shading model is that it only produces opaque material effects, from diffuse to glossy, in direct illumination settings. We believe another approach is required for producing stylized renderings of objects with highly refractive or reflective materials. Indeed, citing Hogarth [1991] (p.120): "The only way to draw [complex refractions] is to observe the seemingly random patterns and record them faithfully." More diffuse effects such as translucency or soft inter-reflections (including color bleeding) may be taken into account by extending the range of shading behaviors. Shadows may also be used to control shading primitives, by mapping occlusion values to primitive parameters. However, we believe that specific shadowing primitives will be necessary to provide an accurate control over shadow appearance.

There are different ways in which our primitive model could be extended in future work. For instance, as demonstrated in supplemental material, we have reproduced highlight shapes of Anjyo et al. [2003; 2006] and Pacanowski et al. [2008]. Although these extensions produce interesting styles, we have preferred retaining simple primitive shapes in our model, mainly because complex shapes do not transfer appropriately to different 3D objects when these have detailed surface features. This is especially noticeable with highlights that are disrupted in the presence of rapidly changing surface curvature. A promising avenue of future work would be to directly control primitive shape on top of 3D objects without any indirection. In particular, such an approach should allow artists to draw highlights that somehow "ignore" small curvature variations to retain their shape during camera, object or lighting motion, while still conveying material properties.

Acknowledgments

This work has been sponsored by the Animare (ANR-08-JCJC-0078-01), SeARCH (ANR-09-CORD-019) projects and the INRIA postdoctoral program.

References

ANJYO, K.-I., AND HIRAMITSU, K. 2003. Stylized Highlights for Cartoon Rendering and Animation. *IEEE Computer Graphics and Applications 23*, 4, 54–61.

ANJYO, K.-I., WEMLER, S., AND BAXTER, W. 2006. Tweakable light and shade for cartoon animation. In *NPAR'2006*, ACM, 133–139.

BARLA, P., THOLLOT, J., AND MARKOSIAN, L. 2006. X-Toon: An extended toon shader. In *NPAR'2006*, ACM, 127–132.

BRUCKNER, S., AND GRÖLLER, M. E. 2007. Style transfer functions for illustrative volume rendering. *Computer Graphics Forum 26*, 3 (Sept.), 715–724.

GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH 98*, ACM, 447–452.

HOGARTH, B. 1991. *Dynamic Light and Shade*. Watson Guptill.

MCCLOUD, S. 1994. *Understanding Comics: The Invisible Art*. Harper Paperbacks.

MITCHELL, J., FRANCKE, M., AND ENG, D. 2007. Illustrative rendering in Team Fortress 2. In *NPAR'2007*, ACM, 71–76.

OKABE, M., MATSUSHITA, Y., SHEN, L., AND IGARASHI, T. 2007. Illumination Brush: Interactive Design of All-Frequency Lighting. In *Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, 171–180.

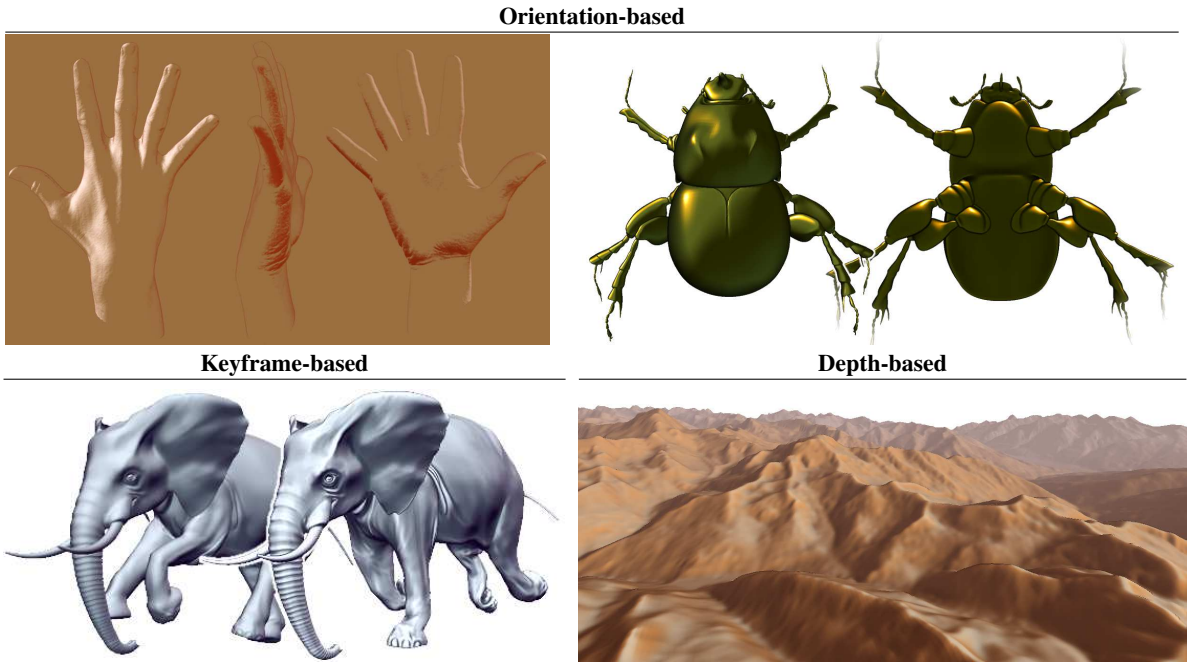


Figure 8: ***Shading controls.*** *Primitive parameters are dynamically modified based on various controls.*

PACANOWSKI, R., GRANIER, X., SCHLICK, C., AND POULIN, P. 2008. Sketch and Paint-based Interface for Highlight Modeling. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 17–23.

RUSINKIEWICZ, S., BURNS, M., AND DECARLO, D. 2006. Exaggerated Shading for Depicting Shape and Detail. *ACM Trans. Graph. 25* (July), 1199–1205.

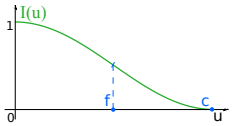
SLOAN, P.-P. J., MARTIN, W., GOOCH, A., AND GOOCH, B. 2001. The lit sphere: A model for capturing NPR shading from art. In *Graphics interface 2001*, Canadian Information Processing Society, 143–150.

VERGNE, R., BARLA, P., GRANIER, X., AND SCHLICK, C. 2008. Apparent relief: A shape descriptor for stylized shading. In *NPAR'2008*, ACM, 23–29.

VERGNE, R., PACANOWSKI, R., BARLA, P., GRANIER, X., AND SCHLICK, C. 2010. Radiance scaling for versatile surface enhancement. In *I3D '10: Proc. symposium on Interactive 3D graphics and games*, ACM, 143–150.

Appendix

The cutoff parameter c corresponds to the angle at which the profile intensity function reaches zero. The bias function $\beta(c)$ gives the bias value that satisfies this condition: $I(c) = 0$. It is defined by



$$\beta(c) = -\frac{\cos(c)}{1-\cos(c)}. \tag{7}$$

The falloff parameter f locates the angle at which the profile function passes through an inflection point, which corresponds to the zero-crossing of its second derivative. The exponent function $\gamma(f,c)$ gives the (strictly positive) exponent value that satisfies this condition: $I''(f) = 0$. The profile's second derivative is:

$$\begin{aligned} I''(x) \quad = \quad & (\beta-1)\gamma\cos x(\beta+(1-\beta)\cos x)^{\gamma-1} + \\ & (1-\beta)^2(\gamma-1)\gamma(\beta+(1-\beta)\cos x)^{\gamma-2}\sin^2 x. \end{aligned}$$

There is only one strictly positive zero-crossing for $I''(f)$:

$$\gamma(f,c) = \frac{\beta - \beta\cos(f) - 1}{(\beta-1)\sin^2(f)}.$$

Replacing β by the formula for the bias function $\beta(c)$ gives:

$$\gamma(f,c) = \frac{1-\cos(c)\cos(f)}{\sin^2(f)}. \tag{8}$$

GLSL shader to compute a primitive contribution. We use a constant color instead of color ramps for simplicity.

```
// inputs from varying, textures or previous pass
vec3 l, n, v, t, b; float kappa;
// shading parameters :
float alpha, tau, beta, gamma, lambda, mu, chi;
// beta(c) and gamma(f, c) are computed on CPU
vec3 color;

float u(vec3 n, vec3 l, vec3 v){
  vec3 h = normalize(l+v);
  float eta = dot(l, v)*.5+.5;
  float S_l = lambda>=0.?1./(1.-lambda)*eta+(1.-eta):
                         1./(1./(1+lambda)*eta+(1-eta));

  vec3 ht = dot(h,t)*t;
  vec3 hb = dot(h,b)*b;
  vec3 hn = dot(h,n)*n;

  h = normalize(S_l*ht + 1/S_l*hb + hn);

  v = reflect(-l, h);
  vec3 r = reflect(-v, n);
  vec3 d = normalize((1.-alpha)*n+alpha*r);

  tau += mu*tanh(kappa*chi);
  return clamp(acos(dot(d, l))-tau,0., PI);
}

float I(float x){
  return pow(max(beta+(1-beta)*cos(x), 0.), gamma);
}

void main(void) {
  gl_FragColor = vec4(color, I(u(l, n, v)));
}
```